

準備

Rについて

R言語は統計解析やグラフ描画に特化したオープンソースの言語です。オープンソースなので将来どこへ行っても使うことができます。オープンソースなのでネットに情報が溢れています。オープンソース最高！また、文法が他の言語よりも簡単でプログラミングを全くやったことのない人でも始めやすい言語です。当研究室では情報量の大きい実験データの定量的、統計的解析が求められることが多いのでRを使えると研究が飛躍的にはかどります。それではまずはRとその統合開発環境であるRstudioのインストールからはじめましょう。

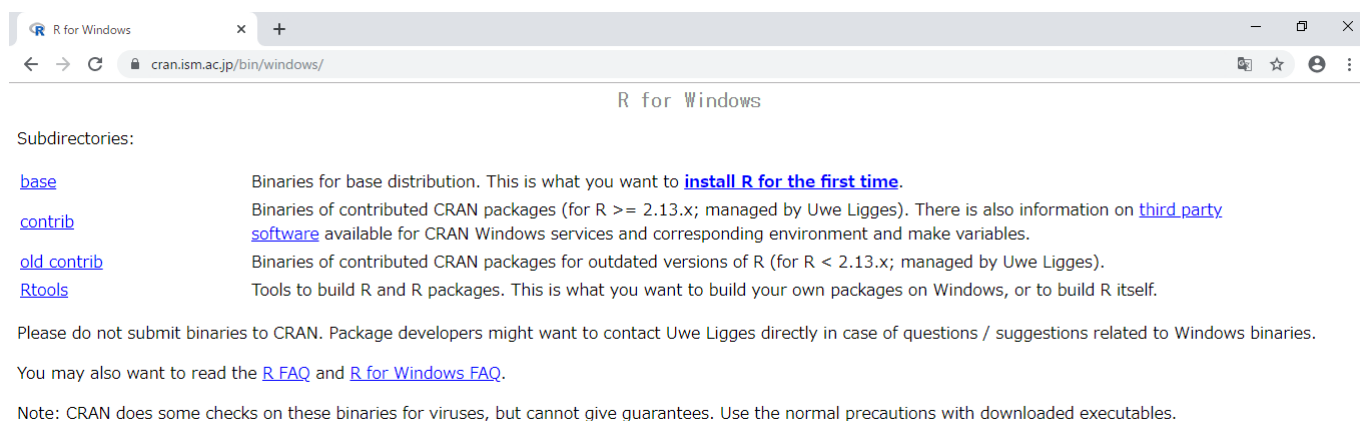
Rのインストール

以下のURLを踏んでください。

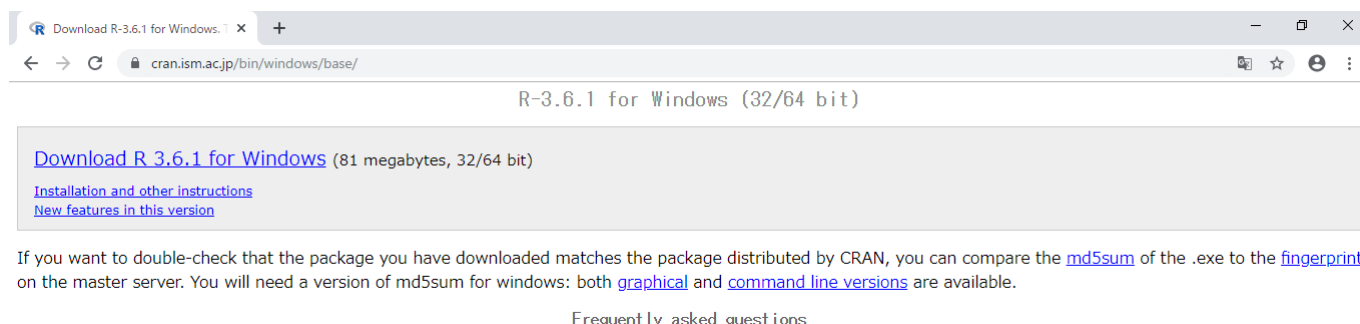
Windows: <https://cran.ism.ac.jp/bin/windows/>

Mac: <https://cran.ism.ac.jp/bin/macosx/>

以下Windows版をインストールしたきのスクリーンショットを表示します。



"install R for the first time" をクリック。



"Download R XXX for Windows" をクリックしてインストーラーをダウンロードして実行する。

"次へ"を押して行ってください。

コンピューターの基本的な情報の表示

Windows のエディション

Windows 10 Home

© 2018 Microsoft Corporation. All rights reserved.

システム

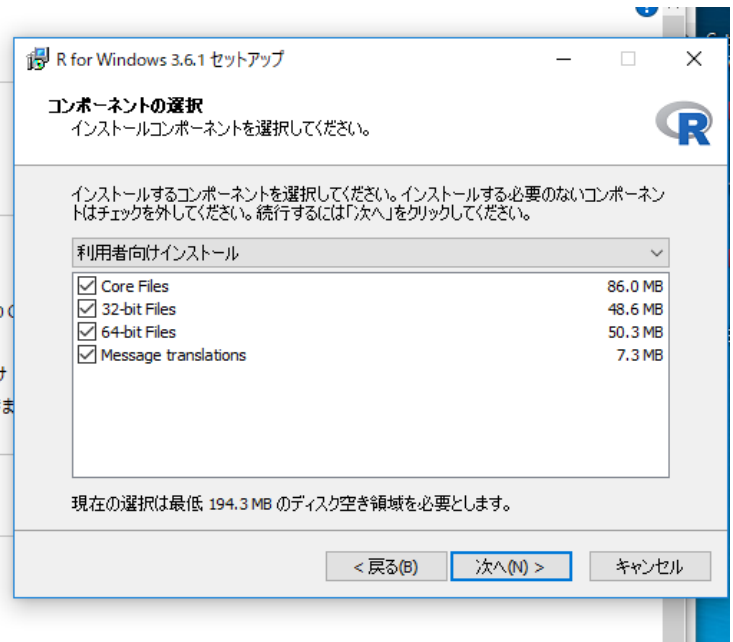
製造元: Dell
 モデル: Vostro 3360
 プロセッサ: Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz 1.80 GHz
 実装メモリ (RAM): 8.00 GB (7.86 GB 使用可能)
 システムの種類: 64 ビット オペレーティング システム, x64 ベース プロセッサ
 ペンとタッチ: このディスプレイでは、ペン入力とタッチ入力は利用できません

Dell サポート

Web サイト: [オンライン サポート](#)

コンピューター名、ドメインおよびワークグループの設定

コンピューター名: ICSLab-PC
 フル コンピューター名: ICSLab-PC
 コンピューターの説明:



この画面のときにOSが32 bitか64 bitかを確認して自分のOS出ない方のチェックを外してください。(自分のOSが64 bitなら"32-bit files" のチェックを外す.)

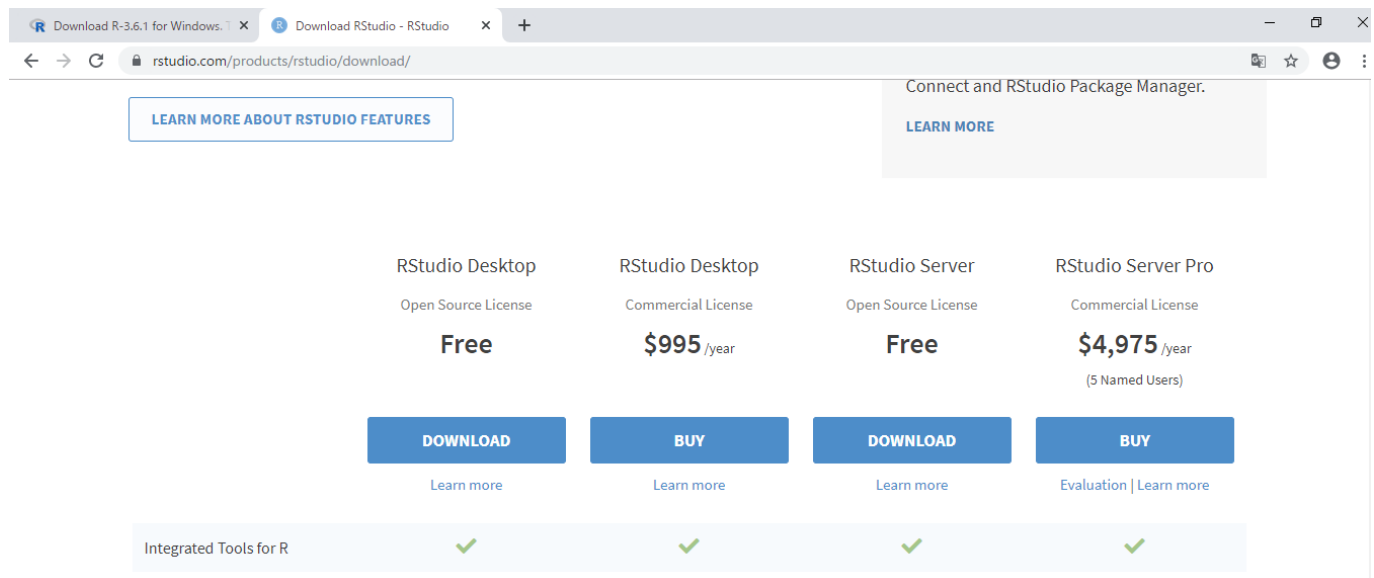
最後まで行けばRのインストールは完了です。

次にRstudioのインストールを行います。

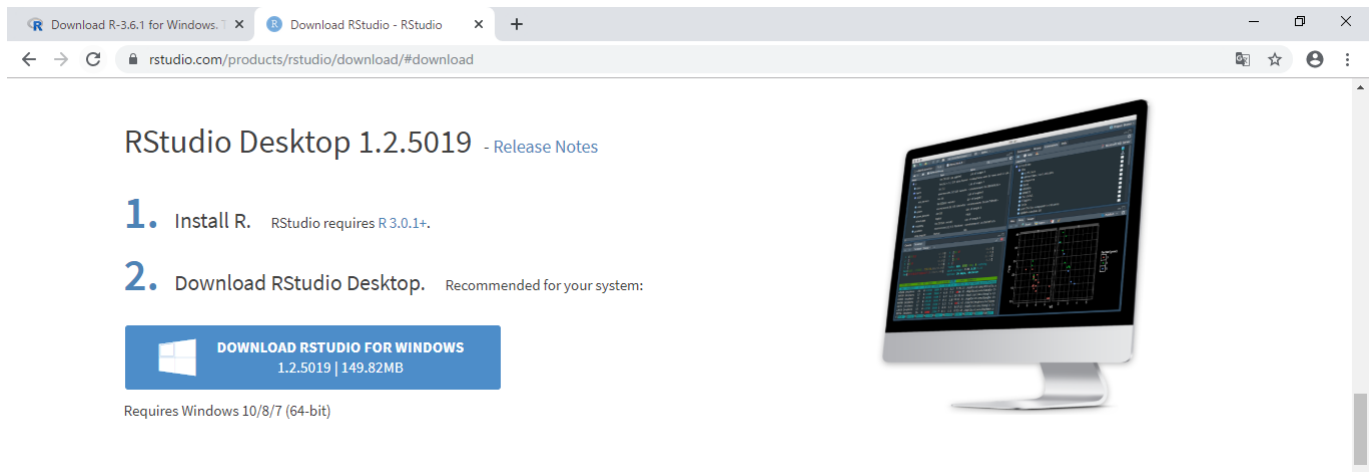
Rstudioのインストール

以下のURLを踏んでください。

<https://rstudio.com/products/rstudio/download/>



一番左の"RStudio Desktop"のFree版の"DOWNROAD"をクリック。



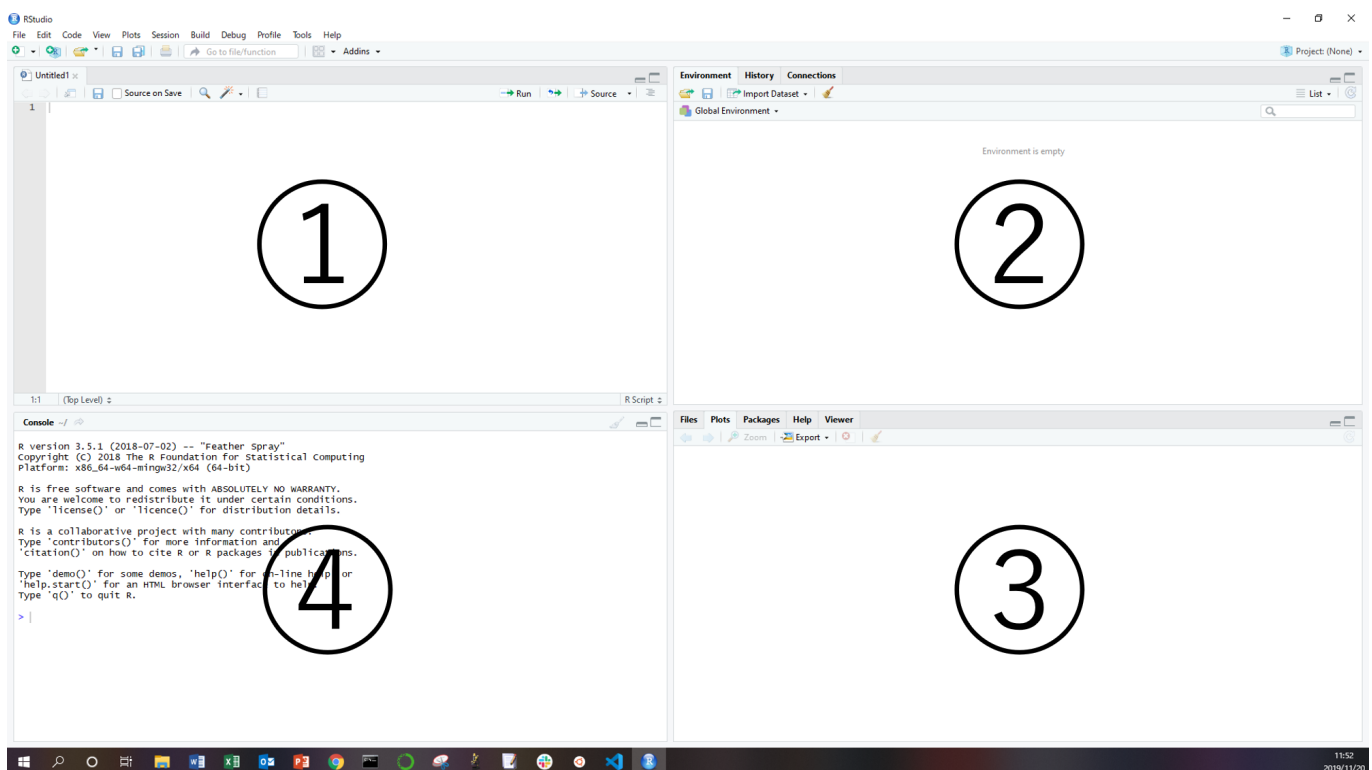
"DOWNROAD RSTUDIO FOR WINDOWS"をクリック.インストーラーをダウンロードして実行. 途中デスクトップにアイコンを作成するか聞かれるのでチェックを入れてください。(忘れてしまった場合は Rstudio/bin/rstudio.exeが実行ファイルなのでこのショートカットをデスクトップなり好きなところに作っておくと便利.)

では、いよいよRを動かしてみましよう。

実際に動かそう！

Rstudioの起動

Rstudioを起動したらctrl + shift + Nを押してください.すると以下のような画面になるはずです。



1. エディタ. ここにスクリプトを打ちます.
2. 環境の表示. ここには定義した変数名とその値がまとめられます.
3. プロット. 描画した図はここに表示されます.
4. コンソール. エラーや警告はここに表示されます.

白い画面は長時間作業をしていると目が疲れてくるので暗いテーマに変更することをおすすめします. 左上のtools→Global Options→Appearance→Editor themeで好きなテーマを選べます.

簡単な計算

それではまず簡単な計算から. 左上のエディタに以下のような計算を打ち込んで全選択してctrl + Enterを押して実行してください.

```
1 + 3 #summation
4 - 2 #subtraction
7/2 #division
8*9 #multiplication
2**4 #power
12%5 #mod
```

#より後ろに書いた文字はコメントとして無視されます. 実行するとコンソールに以下のように結果が表示されます.

```
> 1 + 3 #summation
[1] 4
> 4 - 2 #subtraction
[1] 2
> 7/2 #division
[1] 3.5
> 8*9 #multiplication
[1] 72
> 2**4 #power
[1] 16
> 12%%5 #mod
[1] 2
```

[1]というのは今はあまり気にしなくても大丈夫です.

コードの保存

ctrl + S か file→Save Asで保存できます.

変数

次は変数を使った計算をしてみましょう. 変数とはデータ (値) を格納しておく箱のことです. 以下のコードをエディタに打ち込んで全選択→ctrl + Enterで実行してください.

```
x <- 8
y <- 2
x + y
```

実行結果

```
> x <- 8
> y <- 2
> x + y
[1] 10
```

このコードでは変数x, yにそれぞれ8, 2を代入してからxとyを足し合わせています。Rでは変数に値を格納する（代入と言います。）ときは"<-"または"="を使います。"<-"の方が好まれる傾向があります。変数は以下のように上書きすることができます。逆に間違えて同じ変数名を使ってしまって、望まない上書きがされることのないよう気をつけましょう。

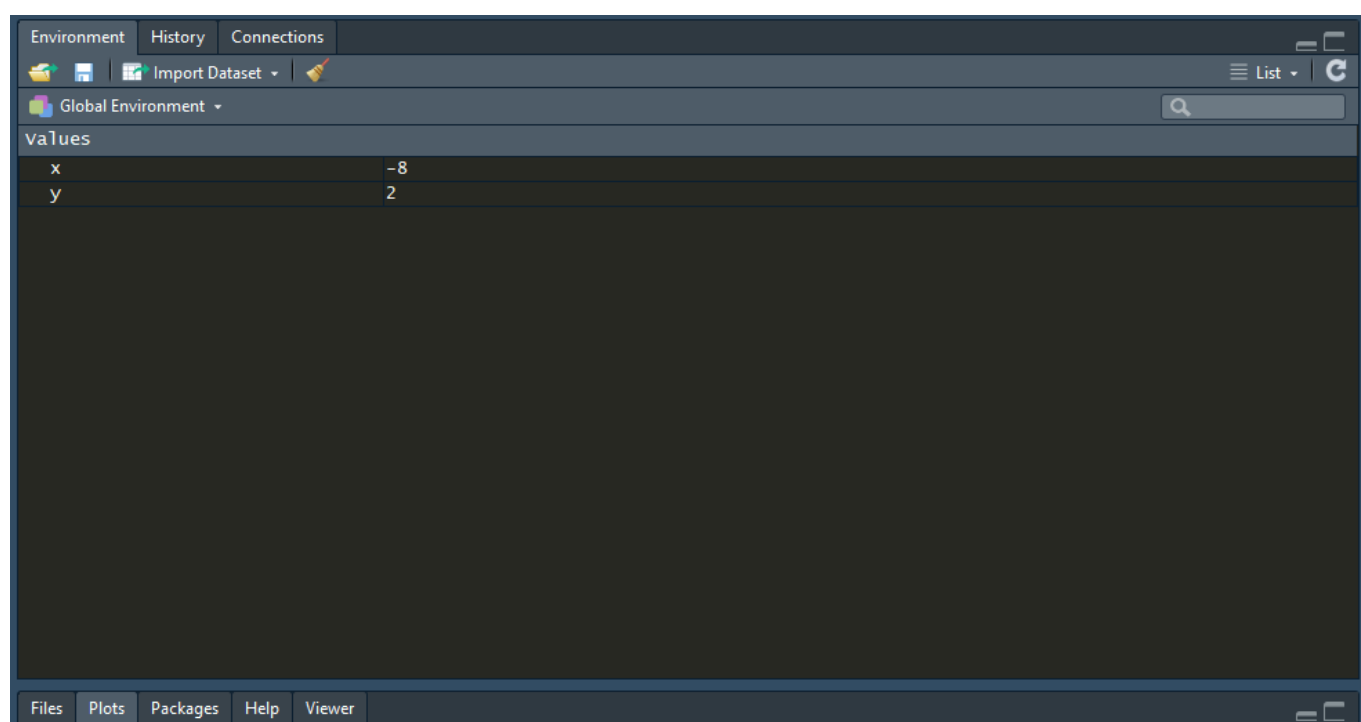
```
x <- 8
y <- 2
x + y
```

```
x <- -8
x + y
```

実行結果

```
> x <- 8
> y <- 2
> x + y
[1] 10
>
> x <- -8
> x + y
[1] -6
```

さて今右上の環境の表示を確認すると以下のように変数名と各変数に格納されている値が表示されています。



もっと複雑な処理を行うようになってきて途中でわからなくなったらこの情報を見直すと良いでしょう。

データ構造

データ構造とはデータを保持する形式のことです。Rのデータ構造で非常によく使うものについて説明します。

ベクトル

- ベクトルは複数の値のインデックス（順番）付きの集合です。以下の例を見てください。

```
vec1 <- c(1,2,3,4)
vec2 <- c(5,5,5,5)

vec1 + vec2 #summation
vec2 - vec1 #subtraction
vec2/vec1 #division
vec1*vec2 #multiplication
vec2**vec1 #power
vec2%%vec1 #mod
```

実行結果

```
> vec1 <- c(1,2,3,4)
> vec2 <- c(5,5,5,5)
>
> vec1 + vec2 #summation
[1] 6 7 8 9
> vec2 - vec1 #subtraction
[1] 4 3 2 1
> vec2/vec1 #division
[1] 5.000000 2.500000 1.666667 1.250000
> vec1*vec2 #multiplication
[1] 5 10 15 20
> vec2**vec1 #power
[1] 5 25 125 625
> vec2%%vec1 #mod
[1] 0 1 2 1
```

Rではベクトルを`c()`で書き表します。数値演算はいわゆる高校のベクトルで習ったルールで上のように行われます。

- ベクトルとスカラーの計算も直感に反することはありません。以下の例を見てください。

```
vec1 <- c(1,2,6,5,8)

vec1 + 5 #summation
vec1 - 5 #subtraction
vec1/3 #division
vec1*3 #multiplication
```

```
vec1**2 #power
vec1%%3 #mod

実行結果
> vec1 <- c(1,2,6,5,8)
>
> vec1 + 5 #summation
[1] 6 7 11 10 13
> vec1 - 5 #subtraction
[1] -4 -3 1 0 3
> vec1/3 #division
[1] 0.3333333 0.6666667 2.0000000 1.6666667 2.6666667
> vec1*3 #multiplication
[1] 3 6 18 15 24
> vec1**2 #power
[1] 1 4 36 25 64
> vec1%%3 #mod
[1] 1 2 0 2 2
```

- ベクトルの要素数は`length()`関数で取得します。関数については後ほど詳しく述べます。

```
vec1 <- c(1,2,6,5,8)
length(vec1)

実行結果
> vec1 <- c(1,2,6,5,8)
> length(vec1)
[1] 5
```

- n番目の要素を取り出したい場合は`[n]`といった記法を使用します。以下の例を見てください。

```
vec1 <- c("c-Fos", "c-Myc", "NFKBIA", "TNFAIP3", "JUN", "JUNB")
vec1[3]

vec2 <- c(4,2,6,8,7,1,6,6,5)
vec2[1]
vec2[4]
vec2[1] + vec2[4]

実行結果
> vec1 <- c("c-Fos", "c-Myc", "NFKBIA", "TNFAIP3", "JUN", "JUNB")
> vec1[3]
[1] "NFKBIA"
>
> vec2 <- c(4,2,6,8,7,1,6,6,5)
> vec2[1]
[1] 4
> vec2[4]
[1] 8
```

```
> vec2[1] + vec2[4]
[1] 12
```

文字列を値として扱う場合は必ず""で囲います。これを忘れてしまうと変数としてみなされてエラーが出ます。

- ベクトルから n 番目の要素を削除したベクトルを作るときは以下のように書きます。

```
vec1 <- c("c-Fos", "c-Myc", "NFKBIA", "TNFAIP3", "JUN", "JUNB")
vec1
n <- 3
vec2 <- vec1[-n] #Delete n th element
vec2
```

実行結果

```
> vec1 <- c("c-Fos", "c-Myc", "NFKBIA", "TNFAIP3", "JUN", "JUNB")
> vec1
[1] "c-Fos"   "c-Myc"   "NFKBIA"  "TNFAIP3" "JUN"     "JUNB"
> n <- 3
> vec2 <- vec1[-n] #Delete n th element
> vec2
[1] "c-Fos"   "c-Myc"   "TNFAIP3" "JUN"     "JUNB"
```

- 連続した整数値のベクトルを作りたいときは $x:y$ のように書きます。

```
vec <- 1:10
vec
```

実行結果

```
> vec <- 1:10
> vec
[1] 1 2 3 4 5 6 7 8 9 10
```

- 初項 x , 末項 y , 公差 d の等差数列をベクトルにしたい場合は`seq()`関数を使って以下のように書きます。

```
x <- 0
y <- 10
d <- 2

seq(x,y,d)
```

実行結果

```
> x <- 0
> y <- 10
> d <- 2
>
```



```
> seq(x,y,d)
[1] 0 2 4 6 8 10
```

- 2つのベクトルをつなげるには以下のように書きます。

```
vec1 <- c("c-Fos", "c-Myc", "NFKBIA")
vec2 <- c("TNFAIP3", "JUN", "JUNB")
vec3 <- c(vec1,vec2)
vec3

実行結果
> vec1 <- c("c-Fos", "c-Myc", "NFKBIA")
> vec2 <- c("TNFAIP3", "JUN", "JUNB")
> vec3 <- c(vec1,vec2)
> vec3
[1] "c-Fos" "c-Myc" "NFKBIA" "TNFAIP3" "JUN" "JUNB"
```

- ベクトル`vec`を n 回繰り返したベクトル`multi_vec`を作りたい場合は`rep()`関数を使って以下のように書きます。

```
vec <- c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
n <- 3

multi_vec <- rep(vec,3)
multi_vec

実行結果
> vec <- c("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat")
> n <- 3
>
> multi_vec <- rep(vec,3)
> multi_vec
[1] "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri"
"Sat" "Sun" "Mon" "Tue"
[18] "Wed" "Thu" "Fri" "Sat"
```

[1]というのはその行の一番左の要素が1番目で[18]というのはその行の一番左に要素が18番目であることを示します。

- ベクトルの中にある特定の要素が何番目にあるかを取得するには例えば`which()`関数を使用します。

```
vec1 <- c("c-Fos", "c-Myc", "NFKBIA", "TNFAIP3", "JUN", "JUNB")
which(vec1 == "TNFAIP3")

vec2 <- c(4,5,6,4)
which(vec2 == 4)
```

実行結果

```
> vec1 <- c("c-Fos", "c-Myc", "NFKBIA", "TNFAIP3", "JUN", "JUNB")
> which(vec1 == "TNFAIP3")
[1] 4
>
> vec2 <- c(4,5,6,4)
> which(vec2 == 4)
[1] 1 4
```

上の例では`vec1`の中の"TNFAIP3"のインデックス（何番目か）と`vec2`の中の4のインデックスを出力しています。==は論理演算子と呼ばれるものでこの場合は「等しい」という意味です。論理演算子についても後ほど詳しく述べます。

データフレーム

次にデータフレームというデータ構造について説明します。データフレームとは以下のような見た目のデータ構造です。

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
11	5.4	3.7	1.5	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
13	4.8	3.0	1.4	0.1	setosa
14	4.3	3.0	1.1	0.1	setosa
15	5.8	4.0	1.2	0.2	setosa

つまり表です。このデータはRに予め用意されているサンプルである`iris`というデータです。これはあやめのがく片(Sepal)と花びら(Petal)それぞれの長さや幅の品種(Species)ごとのデータです。全部で150個の花について調べています。まずは`iris`を変数`df`に格納しましょう。

```
df <- iris
```

- データフレームの行数を取得するには`nrow()`関数を、列数を取得するには`ncol()`関数を取得します。

```
nrow(df)
ncol(df)
```

実行結果

```
> nrow(df)
[1] 150
> ncol(df)
[1] 5
```

- データフレームは\$を使って列をベクトルとして取り出すことができます。以下の例ではdfの`Sepal.Length`列をベクトルとして取り出しています。

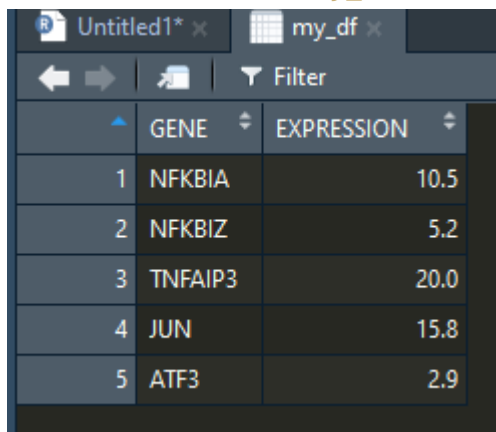
```
vec <- df$Sepal.Length
```

- データフレームをベクトルから自作するには以下のようにdata.frame()関数を使います。

```
gene <- c("NFKBIA", "NFKBIZ", "TNFAIP3", "JUN", "ATF3")
expression <- c(10.5, 5.2, 20.0, 15.8, 2.9)

my_df <- data.frame(GENE = gene, EXPRESSION = expression)
```

これを実行すると環境にmy_dfが追加されるのでクリックして見ると以下のように表示されるはずです。



	GENE	EXPRESSION
1	NFKBIA	10.5
2	NFKBIZ	5.2
3	TNFAIP3	20.0
4	JUN	15.8
5	ATF3	2.9

練習問題

- 1~100までの整数それぞれの2乗の値を出力してください。
- irisのPetal.Width列をベクトルとして取り出し、10番目の要素を削除してください。
- irisのPetal.Lengthが1.4である行数を出力してください。

関数

数学での「関数」の定義をおさらいしましょう。「関数」とは入力を与えられたときに一意に結果を返すものです。飲み物の自動販売機を例に考えましょう。飲み物の自動販売機でお金を入れてボタンを押すと必ずそ

のボタン上に置かれている飲み物と同じ飲み物が落ちてきます。同じボタンを2回おして一回目はコーラが出てきて二回目は青汁が出てくるなんてことはありません。もしそんなことが起きたなら自動販売機の故障なのですみやかに管理会社に連絡しましょう。

プログラミングにおける関数も同じです。ある入力（引数（ひきすう）と言います。）が与えられたときに定義された計算を行って結果を返してくれます。また、関数はほとんどのプログラミング言語で `function(引数)` の形をしています。

Rには様々な便利な計算を行ってくれる関数が用意されています。ここではその中でも特に統計解析に欠かせない関数を紹介します。

ベクトルに関する関数

- `sum()`
ベクトルを引数にとってその総和を返します。

```
vec1 <- c(1,2,6,5,8)
sum(vec1)
```

実行結果

```
> vec1 <- c(1,2,6,5,8)
> sum(vec1)
[1] 22
```

- `mean()`
ベクトルを引数にとってその平均を返します。

```
vec1 <- c(1.2, 1.3, 1.0, 1.5, 1.2, 1.4, 1.3, 1.0, 0.9)
mean(vec1)
```

実行結果

```
> vec1 <- c(1.2, 1.3, 1.0, 1.5, 1.2, 1.4, 1.3, 1.0, 0.9)
> mean(vec1)
[1] 1.2
```

- `sd()`
ベクトルを引数にとってその標準偏差を返します。

```
vec1 <- c(1.2, 1.3, 1.0, 1.5, 1.2, 1.4, 1.3, 1.0, 0.9)
sd(vec1)
```

実行結果

```
> vec1 <- c(1.2, 1.3, 1.0, 1.5, 1.2, 1.4, 1.3, 1.0, 0.9)
> sd(vec1)
[1] 0.2
```

- `median()`

ベクトルを引数にとってその中央値を返します.

```
vec1 <- c(1.1, 1.3, 1.0, 1.5, 1.0, 1.4, 1.3, 1.0, 0.9)
median(vec1)
```

実行結果

```
> vec1 <- c(1.1, 1.3, 1.0, 1.5, 1.0, 1.4, 1.3, 1.0, 0.9)
> median(vec1)
[1] 1.1
```

- `max()`, `min()`

ベクトルを引数にとってそれぞれその最大値と最小値を返します.

```
vec1 <- c(1.1, 1.3, 1.0, 1.5, 1.0, 1.4, 1.3, 1.0, 0.9)
```

```
max(vec1)
min(vec1)
```

実行結果

```
> vec1 <- c(1.1, 1.3, 1.0, 1.5, 1.0, 1.4, 1.3, 1.0, 0.9)
>
> max(vec1)
[1] 1.5
> min(vec1)
[1] 0.9
```

- `which.max()`, `which.min()`

ベクトルを引数にとってそれぞれその最大値と最小値のインデックスを返します.

```
vec1 <- c(1.1, 1.3, 1.0, 1.5, 1.0, 1.4, 1.3, 1.0, 0.9)
```

```
which.max(vec1)
which.min(vec1)
```

実行結果

```
> vec1 <- c(1.1, 1.3, 1.0, 1.5, 1.0, 1.4, 1.3, 1.0, 0.9)
>
> which.max(vec1)
[1] 4
> which.min(vec1)
[1] 9
```

- `sort()`

ベクトルを引数にとってそれを昇順, 降順に並べ替えます.

```
vec <- c(5, 1, 4, 2, 3)

sort(vec) #Ascending order
sort(vec, decreasing=TRUE) #Descending order

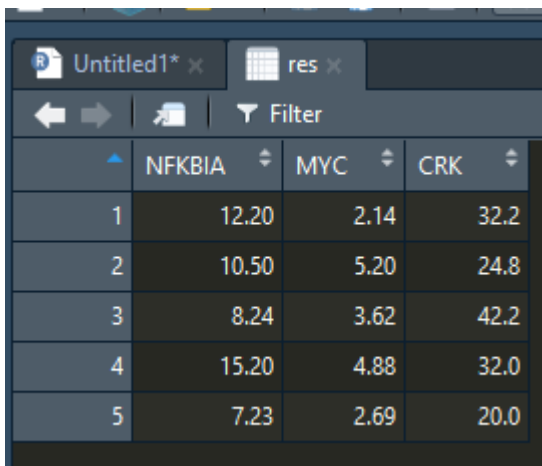
実行結果
> vec <- c(5, 1, 4, 2, 3)
>
> sort(vec) #Ascending order
[1] 1 2 3 4 5
> sort(vec, decreasing=TRUE) #Descending order
[1] 5 4 3 2 1
```

データフレームに関する関数

ここではサンプルデータとして以下のようなデータフレーム`res`を用います。

```
gene1 <- c(12.2, 10.5, 8.24, 15.2, 7.23)
gene2 <- c(2.14, 5.20, 3.62, 4.88, 2.69)
gene3 <- c(32.2, 24.8, 42.2, 32.0, 20.0)

res <- data.frame(NFKBIA = gene1, MYC = gene2, CRK = gene3)
```



	NFKBIA	MYC	CRK
1	12.20	2.14	32.2
2	10.50	5.20	24.8
3	8.24	3.62	42.2
4	15.20	4.88	32.0
5	7.23	2.69	20.0

- `colnames()`
データフレームを引数にとってその列名をベクトルで返す。

```
colnames(res)

実行結果
> colnames(res)
[1] "NFKBIA" "MYC"    "CRK"
```

これを用いて列名の変更も可能である。

```
colnames(res) <- c("Nfkbia", "Myc", "Crk")
colnames(res)
```

実行結果

```
> colnames(res) <- c("Nfkbia", "Myc", "Crk")
> colnames(res)
[1] "Nfkbia" "Myc"    "Crk"
```

- `colMeans()`

データフレームを引数にとってその各列の平均値をベクトルで返す。

```
colMeans(res)
```

実行結果

```
> colMeans(res)
NFKBIA    MYC    CRK
10.674   3.706 30.240
```

データフレームを引数にとってその各行の平均値をベクトルで返す `rowMeans()` もあります。

練習問題

1. `iris`の`Sepal.Length`行の平均値, 標準偏差, 最大値, 最小値を求めてください。
2. `iris`の列名を一行目から順に`Gaku_no_Nagasa`, `Gaku_no_Haba`, `Hanabira_no_Nagasa`, `Hanabira_no_Haba`, `Syurui`に変更してください。

論理式

論理式とはある命題が正しいかどうかを判定する式です。Rでは命題が正しい場合は`TRUE`と表示され正しくない場合は`FALSE`と表示されます。

比較演算子

Rでは比較演算子を以下のように記述します。

記号	意味
<code>==</code>	等しい
<code><</code>	小なり
<code>></code>	大なり
<code><=</code>	小なりイコール
<code>>=</code>	大なりイコール

```
> a <- 4
> b <- 5
>
> a == b - 1
[1] TRUE
>
> b >= a
[1] TRUE
>
> b < a
[1] FALSE
```

論理演算子

Rでは論理演算子を以下のように記述します.

記号	意味
!	否定
&	且つ
	または

```
> a <- 4
> b <- 5
> c <- 2
>
> a != b #a is not equal to b.
[1] TRUE
>
> a == b & b == b #a is equal to b and b is equal to b.
[1] FALSE
>
> a != b | a == c #a is not equal to b or a is equal to c.
[1] TRUE
```

条件分岐
